Blind Spot in Sensor Fusion: Exploiting an Architectural Vulnerability to Hijack and Precisely Control UAVs

Jongsoo Han
Pohang University of Science and Technology
hanjs@postech.ac.kr

Seulbae Kim*
Pohang University of Science and Technology
seulbae@postech.ac.kr

Abstract

To achieve nimble real-time control while remaining robust to error accumulation, PX4 and ArduPilot UAV control pipelines typically combine high-rate IMU-based attitude controls with low-rate GPS-aided state estimation. We show that this foundational design choice introduces a persistent blind spot: injected gyroscope bias immediately alters control outputs yet remains invisible to the state estimator until slow GPS position updates reveal state inconsistencies. Leveraging this architectural vulnerability, we design a black-box reinforcement learning framework that learns to induce trajectory manipulation. Through asymmetric actor-critic training, we derive a highly effective attack policy that, given bias-injection and external observation capabilities, achieves over 85 % success in steering UAVs into a 10 m-radius cylindrical target region, transfers across different autopilot and airframe types without retraining, bypasses a state-of-the-art anomaly detector and built-in failsafes, and transfers to real-world UAVs.

1 Introduction

Unmanned Aerial Vehicles (UAVs) are now an essential component of modern military and civilian operations, supporting a wide range of tasks such as reconnaissance, logistics, and emergency response. This proliferation has elevated UAV security from a theoretical challenge to an urgent real-world necessity, because UAVs perform tasks where failures can lead to life-threatening consequences. As a result, researchers have increasingly focused on understanding how UAVs can be attacked and how such threats can be effectively detected, mitigated, or prevented.

Existing Attacks. Prior work has exploited UAVs through various attack vectors, among which sensor attacks were dominant. For example, GPS spoofing attempts to redirect a UAV by injecting false satellite signals stronger than authentic ones. However, such attacks are trivially detectable by modern sensor fusion systems and built-in failsafe mechanisms [7, 31].

*Corresponding author.

To avoid GPS-based detection and mitigation, attackers have targeted inertial sensors, *e.g.*, a gyroscope, directly. Acoustic injection attacks [12, 44, 45] and laser-based injection attacks [25] can induce controlled signal injections into MEMS inertial sensors, exploiting a physical channel that leaves no digital trace. Building on these injection capabilities, false data injection (FDI) attacks on magnetometers demonstrated that a UAV's heading can be manipulated by carefully crafting magnetometer measurements that evade EKF-based anomaly detection [5]. Each of these approaches, however, suffers from practical limitations. Inertial sensor injection techniques lack the feedback mechanism required for precise trajectory tracking, resulting in either complete destabilization [32] or only coarse directional drifts rather than targeted navigation [45]. Meanwhile, the magnetometer FDI attack [5] strictly requires the UAV to operate in a GPS-denied environment, where it must rely solely on the magnetometer-based heading estimation.

These limitations expose a *fundamental gap between injection capability and strategic exploitation*. While channels for controlled sensor injection exist, no prior work demonstrates targeted steering of UAVs. Existing attacks either produce uncontrolled effects or are functional only under restrictive conditions, such as GPS-denied environments, limiting their real-world applicability. However, practical adversarial scenarios in GPS-enabled environments necessitate steering a UAV to an attacker-defined region rather than merely causing disruption. Targeted steering of a UAV's final position unlocks advanced objectives that cannot be achieved through uncontrolled drift or forced crashes, such as seizing high-value cargo from logistics UAVs, extracting proprietary technology through hardware reverse engineering, or reprogramming the vehicle for adversarial missions. Achieving this level of targeted control in operational environments where GPS and sensor fusion remain active remains an unsolved and critical challenge, and we aim to address it in this work.

An Architectural Vulnerability. We discovered an architectural vulnerability in widely-used UAV control pipelines that can be exploited to enable targeted steering in GPS-

enabled environments. Both PX4 [39] and ArduPilot [36], the two prominent open-source autopilot systems that power diverse platforms (*e.g.*, agricultural [35, 40] and military UAVs [11, 21]), route gyroscope measurements through two pathways that operate at significantly different effective rates: fast updates feed the rate controller almost immediately, whereas bias estimation by the state estimator, the sensor fusion module, depends on much slower GPS updates. This asymmetry creates a latency window in which injected gyroscope bias influences the controller *before* the state estimator.

Moreover, this latency is amplified by the indirect nature of sensor fusion. Since the gyroscope is the only sensor measuring the angular velocity, the fusion cannot directly validate these readings but must infer errors from accumulated position discrepancies. This slows down the estimator's error correction process, creating an extended window where the injected bias influences the controller before being corrected. As a result, an attacker can introduce attitude deviations that accumulate into position drift.

Two Challenges of Exploitation. We assume that attackers are capable of injecting bias and observing UAV states via existing attack techniques [3, 12, 15, 17, 25, 32, 44, 45]. With these capabilities, exploiting the architectural vulnerability to achieve targeted steering presents two technical challenges. First, the attack policy must remain effective against an adaptive estimator. Since the sensor fusion algorithm continuously compensates for sensor errors, a static bias injection would eventually be neutralized, halting the vehicle. Thus, the policy must *dynamically generate biases* that sustain a sufficient difference from the estimator's internal state to induce drift.

Second, the policy faces a *black-box state inference* challenge: to maintain the bias-to-estimation difference, it must predict the estimator's compensation behavior and inject bias that differs from the estimator's current belief. This requires a predictive model capable of inferring the estimator's hidden internal state (*i.e.*, how much bias it has learned), based solely on the externally observable states. However, analytically constructing such a model is intractable, since the UAV's control pipeline is a meticulous integration of cascaded control loops, nonlinear sensor fusion algorithms, and trajectory planning modules whose interactions create a complex, high-dimensional state space that complicates analytical modeling.

Our Approach. We address these challenges by designing a reinforcement learning framework that learns to exploit the architectural vulnerability via adaptive, state-aware gyroscope bias injection. Our approach uses an asymmetric actor-critic architecture [1, 27]; during training, the critic has access to privileged internal states so it can provide rich feedback to the actor, while the actor is trained to operate using only externally observable information. This solves the black-box state inference challenge, making it directly deployable in real-world black-box scenarios. Since the attack must reason about temporal effects, we equip both the actor and critic with Long Short-Term Memory (LSTM) networks, enabling the agent to learn temporal patterns and implicitly model the estimator's learning dynamics. The trained attack policy, called `GAP` (**G**yroscope **A**ttack **P**olicy), continuously yields gyroscope bias sequences that maintain sufficient magnitude to alter UAV trajectories, solving the dynamic bias generation challenge. Moreover, these sequences remain below the detection thresholds of sensor fusion-based anomaly detectors and physics-based invariant checks [6], concealing the root cause of position drift while steering the UAV toward the target.

Results. Under the assumed injection and observation capabilities, GAP achieves high-accuracy steering into attacker-selected target regions, characterizing an upper bound on steering capability rather than a complete end-to-end physical demonstration. Our evaluation demonstrates an 85.1 % success rate in steering UAVs into a 10 m-radius cylindrical target region, outperforming baselines. In addition, our policy is robust to noise, generalizes across autopilots and airframes without retraining, transfers to real-world flights, and evades a state-of-the-art detector in 76 % of successful attacks.

Contribution. Our key contributions are:

- **Architectural vulnerability analysis:** We identify the gyroscope-EKF correction lag and indirect fusion as a fundamental architectural vulnerability in PX4 and ArduPilot, the two prominent open-source UAV autopilots [2].
- **RL-based black-box steering policy:** We design and present a reinforcement learning-based attack policy that relies solely on external observations to achieve targeted position steering while evading anomaly detectors. We validate the attack in real-world flight experiments and successfully redirect a physical UAV into an attacker-defined target region. Demonstration videos are available at `https://tinyurl.com/UAV-Hijack`.
- **Cross-platform generalizability:** We demonstrate that a single policy trained on PX4 seamlessly transfers to ArduPilot across multiple airframe types without retraining, confirming that the attack exploits shared architectural patterns rather than platform-specific bugs.
- **Open Science:** We have released the trained models, code for evaluation, and all datasets used for evaluation at `https://github.com/postech-compsec/GAP`.

2 Background: UAV Control Pipeline

Overview. Both PX4 and ArduPilot, the two prominent open-source UAV autopilot systems [2], rely on a sophisticated, yet common architectural pattern despite implementation differences [34, 38]. The autopilot system integrates data from multiple sensors: an Inertial Measurement Unit (IMU) containing gyroscopes and accelerometers, sampled at 200 Hz, a GPS for absolute positioning updated at 5 Hz to 10 Hz, and a magnetometer for heading reference updated up to 100 Hz.

An autopilot system processes these sensor inputs through four components, as illustrated in Figure 1. The *Sensors* (❶)

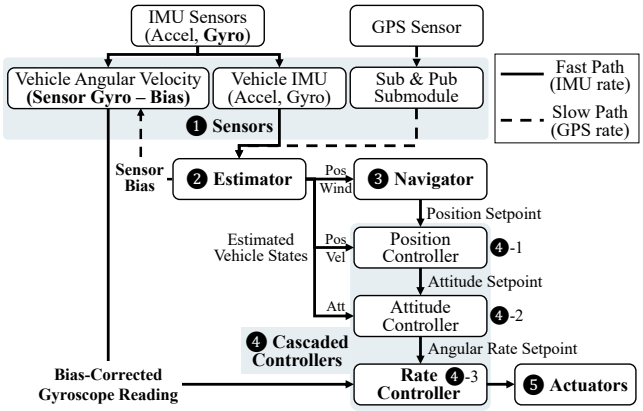


Figure 1: Control pipeline shared by PX4 and ArduPilot.

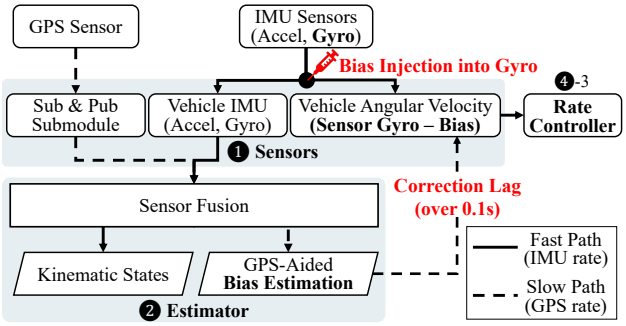


Figure 2: Gyroscope correction lag.

read raw sensor data and publish them to other modules. The *Estimator* (❷) implements sensor fusion, combining sensor data to compute vehicle states. The *Navigator* (❸) generates desired trajectories based on mission objectives. The *Controllers* (❹), via cascaded control loops, emit motor commands to follow the trajectory. The *Actuator* (❺) translates control commands to motor thrusts.

Estimator and Extended Kalman Filter. The Estimator (❷) is implemented using an Extended Kalman Filter (EKF) [30]. It runs at 100 Hz to 200 Hz, fusing multi-sensor measurements to estimate a UAV's states (*i.e.*, the position, orientation, velocities, and sensor biases). To handle varying sensor arrival timings, it synchronizes kinematic states (position, velocity, and attitude) with the IMU's high sampling rate and other states, such as sensor biases (gyroscope and accelerometer) with the GPS's low sampling rate. Most importantly, the estimated sensor biases are published every second. The Sensors module (❶) subscribes to these bias estimates and subtracts them from the raw sensor readings before publishing the corrected angular velocity to the Rate Controller (❹-3).

Indirect Bias Observability. The estimator (EKF) cannot directly observe gyroscope or accelerometer biases because no sensor explicitly measures these biases. It must infer them by detecting inconsistencies between inertial dead reckoning and absolute measurements from external sensors. However, the observability of these biases differs fundamentally. Accelerometer measures linear acceleration, which is what GPS also measures. This enables *head-to-head comparison*; the EKF can immediately detect and learn accelerometer bias by comparing IMU-integrated velocity against GPS velocity measurements, allowing rapid correction when GPS updates arrive. On the other hand, gyroscope measures angular velocity, for which *no other sensor provides a direct measurement*. The EKF cannot directly compare gyroscope readings against any external reference. Instead, gyroscope bias must be *indirectly inferred* by observing discrepancies in downstream states. For example, biased gyroscope produces attitude errors, causing velocity errors that accumulate into position drift. Only by detecting this position drift relative to GPS measurements can the EKF infer gyroscope bias.

Cascaded Control Loops. The Controller module (❹) implements a cascaded control structure with three sequential loops that process control commands from high-level position goals to low-level motor actuation. The *Position Controller* (❹-1), upon receiving the position setpoint from the Navigator (❸), first compares it with the current position from the Estimator (❷) to derive a desired velocity. This is subsequently compared with the current velocity from the Estimator (❷) to generate a desired acceleration. Finally, this acceleration is converted to a desired attitude. The *Attitude Controller* (❹-2) compares the current and desired attitude, producing a desired angular rate. Finally and critically, the *Rate Controller* (❹-3) compares the desired angular rate with the current gyroscope angular rate reading from the sensors module. The rate controller then computes motor thrust commands operating synchronously with gyroscope samples.

3 Preliminary Study

3.1 A Vulnerability Hypothesis

The UAV control pipeline described in §2 is shared across PX4 and ArduPilot. Both route gyroscope-derived body rates into the inner rate-control loop while EKF-level corrections fuse slower external measurements [37, 41]. From this architecture, we hypothesize that the system contains three *architectural vulnerabilities* exploitable as critical attack vectors:

V1. **Fast Path**: The Rate Controller (❹-3) consumes signal-level filtered gyroscope readings, before EKF-level bias correction takes effect. As a result, injected angular velocity bias immediately alters control outputs to Actuators (❺), producing instant physical reactions.

V2. **Slow Path**: The Estimator (❷) attempts to learn and correct sensor biases. However, this process is limited by the arrival rate of GPS updates, which is far slower than that of IMU updates. This creates an inherent delay for the estimator to begin responding to injected errors.

V3. Indirect Observability: As shown in Figure 2, even after GPS readings arrive, the estimator still cannot directly measure gyroscope bias. It must indirectly infer it from physical state discrepancies, making bias estimation intrinsically slow and imprecise.

These observations motivate the following hypothesis: gyroscope bias injection is uniquely capable of exploiting all three architectural vulnerabilities, whereas other sensor injection attacks are not. To validate this hypothesis, we perform the following preliminary experiments.

3.2 Experimental Validation

To validate our discovery of the architectural vulnerability (§3.1), we conducted controlled experiments comparing *gyroscope* and *accelerometer* bias injection attacks' impacts. Our primary objective is to confirm that gyroscope bias injection, unlike other sensor attacks, can effectively exploit the fast path (**V1**) to induce immediate physical deviations, the slow path (**V2**) and the indirect observability (**V3**) to delay correction of these deviations.

Experimental Setup. We implemented a simulated sensor attack environment using PX4 v1.15.4 and the jMAVSim simulator. For the experiment, we utilized custom MAVLink[1] messages to enable real-time bias injection with precise timestamp logging. The simulated UAV was an Iris quadcopter with GPS and IMU sensors. In each trial, the attack was initiated while the UAV hovered at 60 m altitude. We repeated this procedure 50 times per sensor. The GPS update rate in the Software in the Loop (SITL) simulator ran at 20 Hz, higher than the typical 5 Hz to 10 Hz in real deployments.

Since angular velocity and linear acceleration represent fundamentally different physical quantities, direct magnitude comparison of gyroscope and accelerometer attack is infeasible. For fair comparison, we calibrated the two attacks' magnitudes so that their resulting physical impact over the first second of injection is identical. For gyroscope attacks, we injected a bias of 0.06 rad/s on the x-axis and the y-axis, which accumulates to an attitude error of 0.06 rad after one second. This attitude error rotates the gravity vector g where $|g| = 9.8\,\mathrm{m/s^2}$ to produce approximately $0.6\,\mathrm{m/s^2}$ horizontal acceleration (as $g \cdot \sin(0.06) \approx 0.59\,\mathrm{m/s^2}$). For accelerometer attacks, we directly injected a bias of $0.6\,\mathrm{m/s^2}$ on the y-axis, matching the approximate horizontal acceleration produced by the gyroscope attack.

Result 1: Immediate Control Reaction. We first examine the UAV's immediate physical response under each attack type to confirm that gyroscope attack can exploit **V1**.

Figure 3 shows the impact of gyroscope bias injection, averaged over 50 flights. $\dot{\phi}_{est}$ (roll rate estimate, orange line) represents the gyroscope reading used by the controller after

[1]A lightweight, open communication protocol used by UAVs and ground stations to exchange telemetry, command, and sensor data in real time.

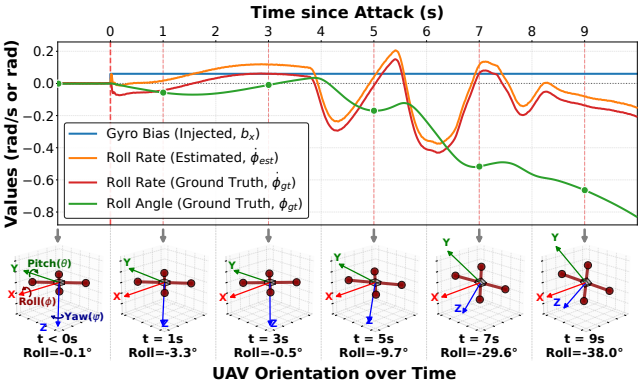


Figure 3: Rate controller's immediate reaction to gyroscope bias, leading to instant tilting of the UAV.

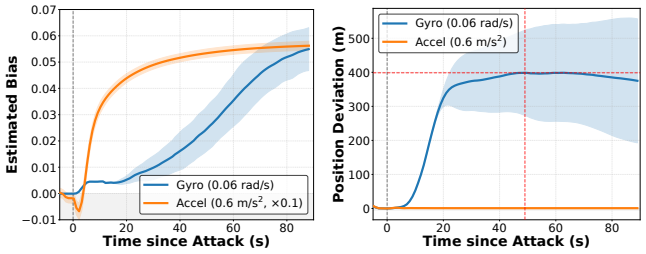
subtracting the currently estimated bias. When a 0.06 rad/s b_x (injected gyroscope bias, blue line) is injected at $t = 0$, the controller interprets this step change as unintended rotation. To maintain the original hover setpoint (*i.e.*, $\phi = 0$), the controller commands an immediate counter-movement to nullify this perceived error. Consequently, $\dot{\phi}_{gt}$ (roll rate ground truth, red line) shifts to -0.06 rad/s, as the Rate Controller (❹-3) applies counter-control to cancel the error. From the controller's perspective, stability appears restored as $\dot{\phi}_{est}$ returns to zero. However, this stability is illusory. To keep the biased sensor reading at zero, the UAV must physically rotate at -0.06 rad/s along the x-axis. This persistent and uncorrected b_x eventually causes the attitude (*i.e.*, ϕ_{gt}: roll ground truth) to diverge continuously, tilting the UAV from -0.1° at $t = 0$ to -38.0° at $t = 9$ as shown in Figure 3 (bottom).

This pipeline-level behavior complements prior Attitude and Heading Reference System (AHRS)-level analysis. Nashimoto *et al.* [23] showed that under low-noise conditions the AHRS Kalman filter heavily weights accelerometer and magnetometer corrections, suppressing gyroscope-only spoofing at the estimator. However, their analysis examines the AHRS in isolation. In the integrated pipeline, the spoofed gyroscope reaches the rate controller via the fast path (**V1**) before estimator-level correction, producing observed tilting.

Result 2: Delayed Estimation Response. To evaluate the exploitability of the rate-limited correction (**V2**) and indirect observability (**V3**), we compare how the EKF estimates bias under gyroscope and accelerometer attacks.

Table 1: Impact comparison of gyroscope (Gyro.) vs. accelerometer (Accel.) attacks. Relative differences highlight gyroscope bias's strength over accelerometer bias.

Metric	Gyro.	Accel.	Relative Diff.
Detection Delay (s)	2.464 ± 0.045	0.138 ± 0.1	17.9× longer
Peak Deviation (m)	404.5 ± 125.4	2.2 ± 0.0	184× larger
Time to Peak Dev. (s)	49.0	3.0	16.3× longer



(a) Bias estimation process **(b)** Position deviation

Figure 4: Comparison of the bias estimation and drift behaviors upon gyroscope and accelerometer attacks.

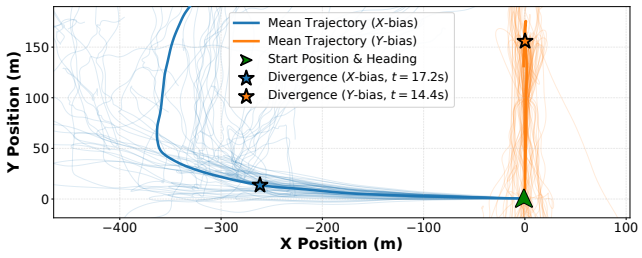


Figure 5: UAV trajectories observed across 50 bias injection trials. Trajectory consistency severely drops beyond the divergence point, marked with stars.

The results are summarized in Table 1. Although both attacks remain unnoticed until the next GPS update (**V2**), the detection delay, defined as the moment when the EKF's bias estimate first changes by more than the publication threshold, differs dramatically; accelerometer bias is detected almost immediately at 0.138 s because the EKF can directly compare IMU-integrated velocity with GPS velocity. In contrast, gyroscope bias detection requires 2.464 s (17.9× longer), since the EKF must wait for downstream position discrepancies to accumulate before inferring angular-rate errors through the indirect observability chain (**V3**). Figure 4(a) visualizes these dynamics. The accelerometer estimate exhibits an initial dip followed by a rapid rise along a steep curve (orange line), reflecting fast and precise estimation. Gyroscope bias estimation, however, begins only after a substantial delay and increases gradually along a gentle slope (blue line), demonstrating the slow and indirect nature of gyroscope bias inference. This confirms the existence of an exploitable window.

Result 3: Divergent Position Dynamics. The estimator's slow response to gyroscope bias attacks also leads to clearly different physical outcomes in the two attacks. As shown in Figure 4(b), the accelerometer attack produces only minor displacement from the initial position. Since the estimator quickly compensates for the injected bias, the peak deviation is only 2.2 m (shown in Table 1). In contrast, the gyroscope attack leverages the indirect fusion pathway, causing the UAV to drift much farther. The displacement continues to increase until it reaches 404.5 m at 49 s (denoted by red dotted lines in Figure 4(b)) at which point the estimator has gathered enough information to start countering the motion.

Result 4: Predictable Trajectory. Finally, we analyze the trajectory patterns across the 50 independent attacks to identify the time interval during which UAV motion remains consistent. The results are summarized in Table 2.

Table 2: Predictable attack trajectory characteristics.

Characteristic	X-axis Attack	Y-axis Attack
Size of Attack Window (s)	17.2	14.4
Mean Position at Divergence (m)	(-260.1, 13.5)	(0.2, 155.8)
Drift Direction	Leftward	Forward

Under identical conditions and identical attacks, the UAV exhibits highly reproducible movement patterns. These repeatable drift patterns arise from how the injected bias affects the controller in the UAV's body frame. The bias is applied on the body-frame axes: the x-axis points forward, corresponding to ϕ (roll), and the y-axis points right, corresponding to θ (pitch). When a bias is injected on the x-axis, the controller interprets it as an unintended right-roll motion and issues a counter-movement by commanding left tilt. This tilt causes the UAV to drift leftward in the global North-East-Down (NED) frame. Similarly, a y-axis bias is interpreted as an unintended pitch change, and the controller compensates by tilting forward, producing forward drift in global coordinates.

As illustrated in Figure 5, across 50 repeated flights with randomized internal states (*e.g.*, EKF covariance and controller setpoints), these counter-movements produce nearly identical early-stage trajectories. This consistency indicates that the UAV's response to bias injection is highly predictable during this initial interval, creating a reliable attack window.

In light of this, we define "attack window" as the period where trajectory standard deviation remains below 10 m, which represents the time during which UAV movements can be reliably predicted. Specifically, for x-axis attacks, this window lasts until 17.2 s, at which point the mean position reaches (-260.1, 13.5) m relative to the hovering position with a standard deviation of 10.2 m. For y-axis attacks, the window closes at 14.4 s with the mean position of (0.2, 155.8) m and a standard deviation of 10.0 m. Beyond these points, trajectory variance increases rapidly as EKF learning progresses at different rates across trials with diverging GPS measurement timing variations and accumulated errors.

Summary. These preliminary experiments validate the architectural vulnerabilities hypothesized in §3.1 and establish the foundation for our attack. **Result 1** confirms that the rate controller reacts instantly to gyroscope bias. **Result 2** shows that slow GPS updates and indirect fusion create a sizable window where injected bias remains uncorrected. **Results 3 and 4** show that UAV motion within this window is highly predictable. Together, these findings expose an exploitable and learnable attack surface.

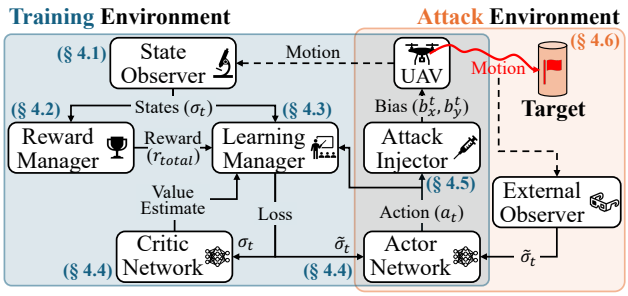


Figure 6: Design and workflow of our reinforcement learning-based gyroscope bias injection attack framework. The trained Actor Network (*i.e.*, attack policy) from the training environment is deployed to attack environment.

3.3 Threat Model

Attacker Capabilities. Building upon the preliminary study, we assume that attackers possess the capability to inject controlled bias into the X and Y axes of the UAV's gyroscope, maintaining a chosen value for one second and updating it at a 1 Hz decision rate. Touch-based acoustic attacks [12] supply this capability via a paster-like unit attached through one-shot physical contact, which drives the gyroscope through solid-medium acoustic coupling during flight, bypassing the attenuation and resonance issues of remote acoustic injection. Gao *et al.* demonstrated this on a Pixhawk4 drone running PX4, sustaining controllable in-flight gyroscope bias of 5 deg/s. Laser-based injection [25], supply chain compromise or sensor driver exploits are additional plausible vectors. We also assume that attackers can observe the UAV's external state (*e.g.*, position and attitude) through visual tracking [3, 15, 17] or publicly broadcast telemetry.

We adopt a realistic *black-box* threat model where attackers cannot read raw sensor values, EKF state, or controller states and setpoints. This reflects typical scenarios where attackers must operate solely on observable UAV behavior.

Attacker Goals. The attacker's primary objective is redirecting a UAV from its current position to an attacker-selected target region (*e.g.*, hundreds of meters away). Success is defined by target-region entry. Arbitrary waypoint-level control and stable terminal stopping are out of scope.

4 Design

The preliminary study in §3 reveals an exploitable window with predictable trajectory patterns. However, deriving deterministic dynamics models to exploit this is infeasible due to nonlinear dynamics of the autopilot stack. We thus propose a closed-loop reinforcement learning (RL) framework where the RL agent autonomously learns modulating gyroscope bias injection to steer the UAV toward a target region.

Overview. Figure 6 illustrates our framework, which first trains an RL model in a training environment and then deploys it to an attack environment. During training, the **State Observer** (§4.1) extracts a 78-dimensional state vector σ_t from the simulator. The **Reward Manager** (§4.2) evaluates σ_t and computes a scalar reward r_{total} that reflects progress toward target-region steering. The **Learning Manager** (§4.3) receives σ_t and constructs two representations: full σ_t for the Critic Network and a partial state vector $\tilde{\sigma}_t$ for the Actor Network. It also computes learning signals such as advantages and loss terms. The **Asymmetric Actor-Critic Networks** (§4.4), both equipped with LSTM layers, process these inputs: the Critic Network consumes σ_t to produce value estimates $v(\sigma_t)$, while the actor consumes only $\tilde{\sigma}_t$ to produce a continuous gyroscope-bias action $a_t = (b_x^t, b_y^t)$. The **Attack Injector** (§4.5) applies this action to the UAV, influencing its motion.

After training, in the **Attack Environment** (§4.6), the trained Actor Network is paired with an External Observer that reconstructs $\tilde{\sigma}_t$ from only the UAV's visible motion, adhering to our threat model (§3.3). This allows the Actor Network to generate bias values and steer the UAV toward the target region in a fully black-box setting.

4.1 State Observer

The State Observer serves as the interface between a UAV's physical dynamics and the learning pipeline. During training, it accesses all internal simulator variables, including:

- **s1**. Normalized time since attack start ($\mathbb{R}^1$)
- **s2**. Relative position from UAV to target region ($\mathbb{R}^3$)
- **s3**. Three-axis linear velocities ($\mathbb{R}^3$)
- **s4**. Three-axis attitude angles ($\mathbb{R}^3$)
- **s5**. Three-axis angular velocities ($\mathbb{R}^3$)
- **s6**. Geometric features ($\mathbb{R}^8$): distance and direction to the target region, approaching velocity, heading error, distance to altitude limits and geofence boundaries, success indicator, and total velocity magnitude
- **s7**. Previous action (a_{t-1}: last injected gyroscope bias, $\mathbb{R}^2$)
- **s8**. EKF bias estimates and variances for gyroscope and accelerometer ($\mathbb{R}^{12}$)
- **s9**. GPS and heading innovation terms and variances ($\mathbb{R}^{14}$)
- **s10**. Controller setpoints ($\mathbb{R}^{12}$): position setpoints, velocity setpoints, attitude setpoints, and angular rate setpoints
- **s11**. Test ratios and position accuracy metrics ($\mathbb{R}^5$)
- **s12**. Navigation state, control and failsafe mode flags ($\mathbb{R}^9$)
- **s13**. Difference between EKF-estimated gyroscope bias and the injected bias ($\mathbb{R}^3$)

The observer converts these into a 78-dimensional state vector σ_t and feeds the Reward Manager (§4.2) and Learning Manager (§4.3), allowing the learning pipeline to track immediate effects of injected gyroscope bias on flight dynamics.

4.2 Reward Manager

The Reward Manager computes the scalar reward signal that guides RL policy learning. Given the full state vector σ_t, it evaluates the UAV's progress toward the target region and computes the total reward

$$r_{\text{total}} = w_d \cdot r_{\text{distance}} + w_v \cdot r_{\text{velocity}} + r_{\text{time}} + r_{\text{termination}} \quad (1)$$

where weights w_d and w_v are configurable[2].

- The **distance reward** $r_{\text{distance}} = (d^{t-1}_{\text{target}} - d^{t}_{\text{target}})$ incentivizes reducing the 3D Euclidean distance d_{target} from the UAV to the target region.
- The **velocity reward** $r_{\text{velocity}} = v_{\text{closing}}$ encourages approaching the target with positive closing velocity, where v_{closing} is the velocity component toward the target. These distance and velocity rewards provide dense feedback that helps the actor discover effective policies even when success is sparse, acting as a hint to guide exploration.
- The **time penalty** $r_{\text{time}} = -0.01$ encourages faster convergence, preventing the actor from learning inefficient attack strategies that require excessive mission duration.
- The **terminal reward** $r_{\text{termination}}$ encodes success and failure conditions. When the UAV enters the target region, the agent receives a reward of $+20$, while timeout, altitude violation, or excessive deviation from the target each incur a penalty of -20. These terminal conditions are continuously evaluated to ensure that brief or transient entries into the target region are correctly detected.

This reward structure balances dense feedback that accelerates learning with sparse terminal rewards for mission success.

4.3 Learning Manager

The Learning Manager coordinates the entire training loop. Upon receiving the full 78-dimensional σ_t from the State Observer, it constructs the partial observation $\tilde{\sigma}_t$ used by the actor. This 23-dimensional vector consists of 21 externally observable quantities (**s1-s6** in §4.1) along with the previous action **s7**: $a_{t-1} = (b_x^{t-1}, b_y^{t-1})$.

In addition to producing state vectors, the Learning Manager computes learning signals, including advantages and loss terms for both Actor and Critic Networks. It then updates both networks through backpropagation of the combined policy and value losses.

4.4 Asymmetric Actor-Critic Networks

Asymmetric Architecture. To satisfy a realistic threat model (§3.3) while maintaining efficient training performance, we adopt an asymmetric actor-critic architecture [1, 27] that separates privileged and non-privileged information pathways. During training, the **Critic Network** receives the full 78-dimensional state vector σ_t, enabling it to compute accurate value estimates and provide informative gradients that accelerate learning. However, the **Actor Network** receives only the 23-dimensional partial state vector $\tilde{\sigma}_t$ that an attacker can realistically construct from external sensing.

LSTM Layers. To further enhance the agent's capability to exploit the temporal attack window (14 s to 17 s depending on axis identified in §3.2), both networks incorporate Long Short-Term Memory (LSTM) layers [14]. This recurrent architecture allows the policy to capture the temporal dynamics of the estimator's gradual bias compensation and the delayed physical response, enabling the agent to reason about *when* to modulate bias injection intensity.

Bounded Actions. The actor ultimately outputs a continuous two-dimensional gyroscope bias **action** $a_t = (b_x^t, b_y^t)$, where each dimension is constrained within $\pm 0.06\,\text{rad/s}$. These bounds were selected to preserve stealthiness while providing sufficient controllability, based on PX4's bias detection thresholds. Its estimator triggers a failsafe when the gyroscope bias exceeds $0.15\,\text{rad/s}$ ($\approx 8.59\,^\circ/\text{s}$), identifying it as a sensor malfunction. Thus, we restrict the attack bias to $0.06\,\text{rad/s}$, providing a $2.5\times$ safety margin below the detection threshold.

4.5 Attack Injector

The Attack Injector applies the actor-generated bias values to the UAV's gyroscope measurements. For each decision step, it modifies the IMU data stream by adding the continuous bias vector to the UAV's angular-rate readings. This perturbation propagates through the UAV's sensor-fusion stack and cascaded control loops, leading to trajectory deviation. This injection capability is grounded in known methodologies. Physical channels such as acoustic [12, 45] and laser-based [25] injection can impose controlled bias on MEMS gyroscopes without requiring software or network access to the UAV; supply chain compromise or sensor driver exploits represent additional system-level vectors. Critically, the attacker needs only to add a bounded bias to the gyroscope stream, with no access to flight commands or internal states. Such channels can maintain the bias for several seconds, meeting the 1 s hold duration required by our 1 Hz decision rate.

4.6 Attack Environment

In the deployment environment, only the trained Actor Network, *i.e.*, the actor policy, is used. Because the attacker has no access to internal UAV states, an External Observer reconstructs the $\tilde{\sigma}_t$ used by the actor. This observer relies solely on the physically observable quantities, such as motion and attitude, inferable from external sensing. Like in the training environment, the actor consumes $\tilde{\sigma}_t$ and outputs a_t, which the Attack Injector (§4.5) applies to the target UAV. The injection loop repeats continuously, enabling the attacker to steer the UAV toward the target region in real time despite having no visibility into the internal estimator states or controller logic.

[2] We empirically tuned the weights to $w_d = 0.1$ and $w_v = 0.025$.

5 Training Strategy

We adopt a stage-wise training approach to achieve the final objective: guiding the UAV into a 10 m-radius target region from an initial distance of 220 m with over 80 % success rate.

Initial Conditions. In all training episodes, the target region is fixed at the origin, and a UAV is initialized in position-hold mode at a random angular position on a 220 m-radius circle around the origin at 60 m altitude (see Figure 7). We implement domain randomization [19, 26, 43] by varying this initial spawn position while fixing the UAV's initial heading to North. This setup is geometrically equivalent to placing the UAV at the origin and commanding it to move toward the target from all possible relative directions in its body frame, ensuring the policy learns omnidirectional control rather than overfitting to specific maneuvers (*e.g.*, only flying forward).

The 220 m starting distance serves as a representative standard for validating long-range robustness; success at this range implies that shorter-range attacks are readily achievable, while longer-range attacks can be constructed by chaining multiple segments as sequential waypoints.

Intermediate Policy: Learning Coarse 3D Control. Training begins with a 20 m-radius *spherical* target region. This geometry offers uniform feedback across all three axes, allowing the agent to first acquire coarse 3D stabilization skills. Especially, the spherical region provides dense, well-shaped reward signals during early exploration, enabling the Actor-Critic Networks to learn how to regulate the UAV's trajectory without diverging. We ran this stage until the success rate exceeded 80 %, which required approximately 10K iterations.

Final Policy: Refined Horizontal Control. After establishing robust coarse control, we transition into the final objective: a *cylindrical* target region with a 10 m-radius and a relaxed 0 m to 120 m altitude tolerance. This transition is motivated by the physical characteristics of the gyroscope bias attacks; injected bias provides strong horizontal steering controls through accumulated attitude deviation, but its influence over the vertical motion is limited, which is primarily governed by throttle control. The cylindrical geometry is well-aligned with the true controllable dimensions of the attack: it demands high horizontal precision, while avoiding unrealistic vertical constraints that the attack cannot satisfy.

We transferred the learned weights from the intermediate policy and continued training with the new cylindrical objective. This curriculum learning strategy [4, 22] allows the actor to reuse the stabilization and trajectory-shaping capabilities acquired during the initial training, while refining horizontal accuracy for the final objective. The resulting policy exceeded 80 % success rate within 6K iterations.

We refer to the final attack policy as `GAP` (**G**yroscope **A**ttack **P**olicy).

6 Implementation

Configuration. We implement the training environment of our framework by integrating the RL architecture from §4 with a high-fidelity UAV simulation environment. The UAV runs an unmodified PX4 autopilot with default estimator and controller configurations. We focus on off-the-shelf multicopter UAVs that rely on IMU, magnetometer, and GPS-based state estimation, with vision-based localization (e.g., VIO) and camera-assisted navigation out of scope. All sensors (GPS, accelerometers, and magnetometers) operate normally, except the gyroscope stream, where the Attack Injector (§4.5) injects bias values produced by the Actor Network (§4.4).

Attack Injector. As defined in §3.3, our threat model assumes controlled gyroscope bias injection via any physical channel. In our implementation, we emulate this capability via a custom MAVLink message, used solely for deterministic and reproducible simulation where physical channels (e.g., acoustic, optical) cannot be faithfully reproduced. The injected values are applied at the sensor driver prior to EKF fusion, emulating the effect of a physical signal injection. This design choice follows standard evaluation practice in cyber-physical security, where hardware-triggered effects are emulated through software injection to ensure repeatability.

Training Infrastructure. Our implementation depends on Python 3.10.18, PyTorch 2.8.0, and Ray RLlib 2.48.0 [18, 46], a scalable distributed RL framework, with the APPO (Asynchronous Proximal Policy Optimization) algorithm [20]. Our custom `AsymmetricLSTMModule` extends its `TorchRLModule` base class to support separate Actor and Critic Networks with distinct observation spaces.

Training employs 30 distributed workers (each running a PX4 instance in jMAVSim simulator with 1 CPU core and 0.1 GPU, distributed across 1 RTX 4060 Ti and 2 Tesla V100) and a learner with 4 CPU cores and 1 RTX 4090 GPU.

Training Environment Details. We bridge our RL framework to PX4 v1.15.4's Software-in-the-Loop firmware via MAVLink protocol. By default, the state estimator and the Rate Controller operate at 250 Hz in the simulator.

The environment launches PX4 and jMAVSim instances per parallel worker, injects gyroscope bias via a custom MAVLink message, receives ground truth kinematic states and privileged information. All simulations run at 4× the wall clock speed. During training, we use ground truth states for the actor's observation (assuming ideal external tracking) and disable external aerodynamic disturbances for stable learning.

Network Architecture Details. The **Actor Network** comprises a 3-layer fully-connected encoder with 512 hidden units per layer, followed by a 2-layer LSTM with 512 hidden units per layer. All fully connected layers use ReLU activation and layer normalization. The **Critic Network** uses a 4-layer, fully-connected encoder with 1024 hidden units per layer followed by a 2-layer LSTM with 1024 hidden units, reflecting the

critic's need to model complex EKF convergence dynamics from privileged information while the actor operates under partial observability with a simpler decision space.

Hyperparameters. Key RL hyperparameters: learning rate $\alpha = 5 \times 10^{-5}$, gradient clipping $\tau_{\text{grad}} = 0.5$, discount factor $\gamma = 0.995$, PPO clip ratio $\varepsilon = 0.2$, entropy $\beta = 0.02$, and value loss $c_v = 1.0$. We use V-trace clipping thresholds $\rho_{\text{clip}} = \bar{\rho}_{\text{clip}} = 1.5$ for off-policy stabilization, with batch size 4096 timesteps, 64-timestep rollout fragments per worker, 1024-timestep minibatches, and 2 epochs per iteration.

7 Evaluation

In this section, we evaluate the effectiveness of our attack framework by answering the following research questions:

RQ1. Can our attack policy (GAP) successfully steer UAVs to a desired position, compared to baselines? (§7.1)
RQ2. Is GAP robust to tracking and injection noises? (§7.2)
RQ3. Is GAP transferable to heterogeneous autopilot platforms and airframes without retraining? (§7.3)
RQ4. Can GAP bypass existing anomaly detectors? (§7.4)
RQ5. Is GAP resilient to built-in failsafe mechanisms? (§7.5)
RQ6. Is the proposed attack feasible in real-world flight missions under environmental disturbances? (§7.6)

Experimental Setup. Unless stated otherwise, all experiments use PX4's default configurations under calm conditions (no wind) with inherent sensor noise enabled for realistic EKF behavior. During evaluation, state observations are derived from the simulator's ground truth kinematic outputs, assuming that an attacker is equipped with external tracking capabilities.

Episode Initialization. To rigorously evaluate the policy's omnidirectional adaptability, UAVs spawn on the circumference of a 220 m-radius circle centered on the target, at a fixed altitude of 60 m. We generate 12 equally spaced spawn locations (every 30°) to verify, with deterministic and uniform coverage of all approach directions, whether the domain-randomization-trained policy generalizes without overfitting to specific trajectories. Episodes timeout upon safety violations or upon reaching a maximum duration of 300 s.

Attack Target Region. To comprehensively assess control precision, we report the attack success rate for the following four target regions: 20 m-radius cylinder and sphere, and 10 m-radius cylinder and sphere. Unless otherwise stated, the "success rate" is measured on the 10 m cylinder target.

Demo Video. The video recordings of our experiments, including the attack on real-world UAVs (§7.6), are available at https://tinyurl.com/UAV-Hijack.

7.1 RQ1: Effectiveness of Trained Policy

We evaluated the effectiveness of GAP (the final policy acquired in §5) by comparing it against three baseline attacks: random bias injection, directional bias injection, and adaptive bias injection. Random bias injection represents a non-strategic adversary who injects gyroscope bias values uniformly sampled from the range $[-0.06, 0.06]$ rad/s into the roll (x) and pitch (y) axes, resampled and updated at a frequency of 1 Hz. Directional bias injection applies a static, feedback-free strategy: based on the UAV's initial spawn angle θ relative to target ($\theta = 0°$ for North, increasing clockwise), it injects a fixed bias vector $b_x = 0.06 \cdot \sin(\theta)$, $b_y = -0.06 \cdot \cos(\theta)$ to induce a persistent drift toward the center. Adaptive bias injection extends these baselines by recalculating the bias vector at 1 Hz, based on the UAV's current position and heading, representing the strongest baseline.

Results. We conducted a comparative evaluation consisting of 24 episodes for each baseline (2 trials at each of the 12 spawn positions) and a large-scale test of 1,200 episodes for our attack policy to ensure statistical robustness. As summarized in Table 3, our policy achieved an 85.1 % success rate on the 10 m cylinder target, outperforming all baselines.

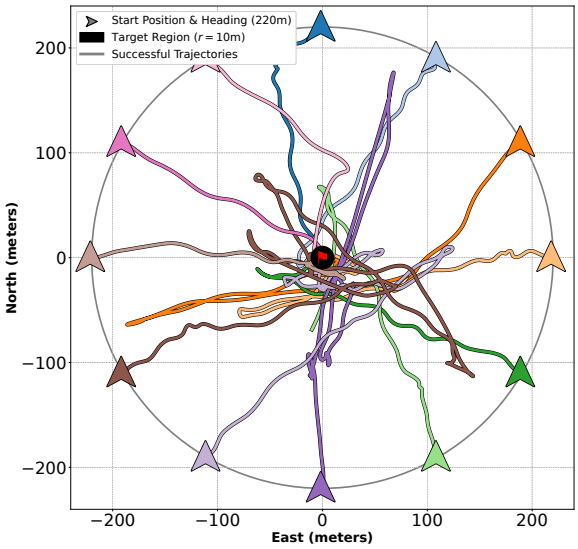
Random bias failed to induce consistent drift, while directional bias allowed the estimator to accumulate error corrections, causing the trajectory to diverge before reaching the target. Adaptive bias reached only 20.8 % on the 10 m cylinder target, as it could not model the EKF's internal compensation dynamics. Among the cases where adaptive bias succeeded, GAP reached the target 64 % faster on average (54.5 s compared to 151.3 s on the 10 m cylinder), further demonstrating the efficiency of GAP.

Figure 7(a) visualizes representative successful trajectories from all 12 spawn positions, showing consistent steering across varying approach angles.

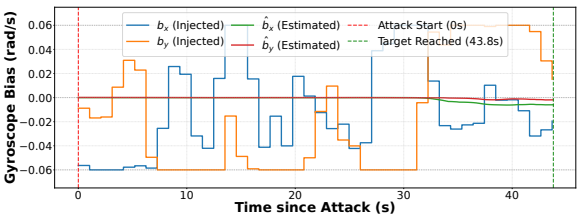
Anatomy of a Successful Attack. Figure 7(b) illustrates the policy's exploitation of the UAV's dynamics in a West-spawned episode. To reach the target at the center, the UAV must drift rightward. Accordingly, the agent initially sustains a maximum x-axis bias of -0.06 rad/s for 7 s. Physically, injecting this negative bias deceives the controller to trigger a counter-acting rightward roll command that drives the vehicle toward the target. This sustained injection is critical: it forces the UAV to tilt significantly, overcoming the vehicle's initial inertia and position-hold stabilization. Once the UAV gains sufficient momentum, the agent switches to a dynamic modulation pattern, updating the bias every second. This behavior demonstrates that the RL policy has learned to leverage the predictable attack window (14 s to 17 s) identified in §3.2.

Table 3: Baseline vs. Our RL-based attack comparison.

Attack Method	Success Rate (%)			
	Cyl. 20 m	Sph. 20 m	Cyl. 10 m	Sph. 10 m
Random Bias	0.0	0.0	0.0	0.0
Directional Bias	4.2	4.2	0.0	0.0
Adaptive Bias	41.7	29.2	20.8	8.3
GAP	93.2	85.0	**85.1**	59.1



(a) Successful trajectories into a 10 m-radius cylinder



(b) An example of state-aware gyroscope bias injection

Figure 7: Analysis of GAP's successful attack cases.

Underlying these decisions, a key requirement is maintaining a discrepancy between the injected bias (b_x and b_y) and the EKF's estimated gyroscope bias($\hat{b}_x$ and $\hat{b}_y$). Once the EKF fully learns the injected bias, the bias-corrected angular rate input to the controller is neutralized, eliminating the induced drift. As shown in Figure 7(b), both $\hat{b}_x$ and $\hat{b}_y$ remain close to zero throughout the episode, suggesting that GAP keeps the estimator from converging to the injected bias.

7.2 RQ2: Robustness to Realistic Noise

Our threat model assumes that the attacker observes the target UAV state via an external vision-based tracker, which tends to be noisy. To further test the robustness of our policy against a realistic attacker, we corrupted the simulator UAV state before feeding it to GAP by adding two types of noises:

Noise source 1: Tracking. We derived noise parameters from Li *et al.* [17], which evaluates 6-DoF pose estimation of a quadrotor at 20 m to 52 m using a telephoto lens. Under their metric, they report that 87.49 % of position estimates have errors below 0.707 m, and 7.42 % of attitude estimates have errors below 0.707°. We modeled per-axis Gaussian noise for position and attitude as $\Delta p \sim \mathcal{N}(0, \sigma_p^2)$ and $\Delta\theta \sim \mathcal{N}(0, \sigma_\theta^2)$. Matching the reported acceptance probabilities yielded $\sigma_p = 0.295$ m and $\sigma_\theta = 1.033°$. Since velocity is estimated as the difference of two consecutive position observations, its noise variance doubles; at the 1 Hz policy sampling interval this gives $\sigma_v = \sqrt{2}\sigma_p/\Delta t = 0.417$ m/s and $\sigma_\omega = \sqrt{2}\sigma_\theta/\Delta t = 0.026$ rad/s. These noises were applied to the states fed to GAP.

Noise source 2: Injection delays and losses. We modeled injection-side imperfections based on our real-world flight logs obtained in §7.6: bias injection command loss with 12.5 % probability, and bias injection period jitter sampled from $\max(1, \mathcal{N}(1.087, 0.076^2))$ s, both derived from measurements across seven flight logs.

Results. Table 4 summarizes results over 1,200 evaluation episodes for each noise source. GAP remained effective under all criteria, demonstrating robustness against tracker-induced uncertainty as well as injection-side delays and losses, without any noise-aware retraining. These results stemmed from three factors. First, injection delays were inherent during training, as the RL framework and the PX4 ran as separate processes communicating over MAVLink, naturally exposing the policy to timing imperfections. Second, because the policy recalculated a new bias based on the newly observed state at each step, missed injections were compensated for at the subsequent decision step. Lastly, robustness to tracking noise was consistent with the known properties of recurrent policies under noisy observations [14]. These noise sources approximate key real-world imperfections but do not cover all aspects of a physical deployment, such as residual bias from injection hardware or distance-dependent tracking degradation. We further discuss this gap in §8.

7.3 RQ3: Cross-platform Generalizability

To show the generalizability of our attacks, we extended our evaluation to a different autopilot platform (ArduPilot) with diverse airframe types, and to a different simulator (*i.e.*, PX4 with Gazebo simulator).

We deployed GAP, which was originally trained on PX4 and jMAVSim simulator, directly to these two configurations *without* any retraining. We conducted 24 evaluation episodes per configuration, consisting of two independent trials at each

Table 4: GAP success rates under tracking noise and injection imperfections. 'None' (no noise) results are from RQ1, and 'Both' condition is when both noise sources were present.

Noise Condition	Success Rate (%)			
	Cyl. 20 m	Sph. 20 m	Cyl. 10 m	Sph. 10 m
None	93.2	85.0	85.1	59.1
Tracking Noise	95.3	89.9	89.2	67.3
Delay & Loss	94.9	87.4	87.3	64.2
Both	95.4	88.1	87.2	63.9

of the 12 equally-spaced spawn positions.

Results. Table 5 shows the success rates across 9 ArduPilot airframes and the PX4 Gazebo environment. The policy achieved 70.8 % to 87.5 % success rates on all multirotor configurations except coaxcopter. This successful transfer confirmed that our attack exploits a fundamental architectural vulnerability (§3.2) rather than platform-specific implementation bugs. In the case of coaxcopter, the success rate dropped to 54.2 % because its control pipeline significantly differed from the common pipeline that we identify in §2.

7.4 RQ4: Detection Evasion

We evaluated the stealthiness of `GAP` against `CI` [6], a powerful anomaly detector that identifies attacks by checking physics-based control invariants, *i.e.*, the correlation between sensor measurements and control commands. We utilized their VMware image, which runs ArduPilot v3.4.

Results. We conducted 24 attack episodes, consisting of two independent trials at each of the 12 equally-spaced spawn positions. The attack achieved an 87.5 % success rate on the 10 m cylinder target, with `CI` detecting only 23.8 % of these successful episodes. Notably, detections occurred exclusively at specific approach angles (3, 4, and 8 o'clock directions), with zero detections in all other orientations, showing that our attack effectively exploited blind spots in the invariant relationships modeled by the detector.

7.5 RQ5: Resilience to Failsafe Mechanisms

The attack experiments in RQ1 (§7.1), also revealed that our attack is robust against built-in failsafe mechanisms. Our observations can be summarized as follows:

Observation 1: Gyroscope Failsafes are Bypassed. Across all PX4 experiments, the gyroscope bias failsafe was *never* triggered. This indicates that the injected bias consistently remained below the platform's detection thresholds, preventing the autopilot from attributing the resulting anomalies to gyroscope sensor corruption.

Table 5: Success Rate Across Platforms and Airframes.

Platform	Airframe	Success Rate (%)			
		Cyl. 20 m	Sph. 20 m	Cyl. 10 m	Sph. 10 m
PX4	x500 (Gazebo)	91.7	70.8	83.3	33.3
ArduPilot	coaxcopter	75.0	62.5	54.2	37.5
	dodeca-hexa	87.5	83.3	70.8	62.5
	hexa	87.5	87.5	79.2	79.2
	octa	91.7	91.7	75.0	66.7
	octaquad	91.7	91.7	87.5	83.3
	quad	87.5	83.3	79.2	75.0
	singlecopter	95.8	95.8	83.3	79.2
	tri	83.3	79.2	79.2	79.2
	y6	95.8	91.7	79.2	70.8

Observation 2: Our attack ignores other failsafes. In 20.3 % of attack episodes, attack-induced drift triggered generic failsafes (*e.g.*, "invalid local position" and "invalid local velocity"), causing the autopilots to initiate safety behaviors such as Landing or Return-to-Launch (RTL). However, entering these modes did not neutralize our attack; in every case, the triggered failsafe was immediately deactivated within an average of 2.1 s and our attack continued to steer the UAV to the target region. This demonstrates that even when failsafe logic engages, it is fundamentally ineffective against our gyroscope-based attacks.

7.6 RQ6: Sim-to-Real Validation

Finally, we evaluate whether `GAP` transfers to physical flight through real-world flight experiments.

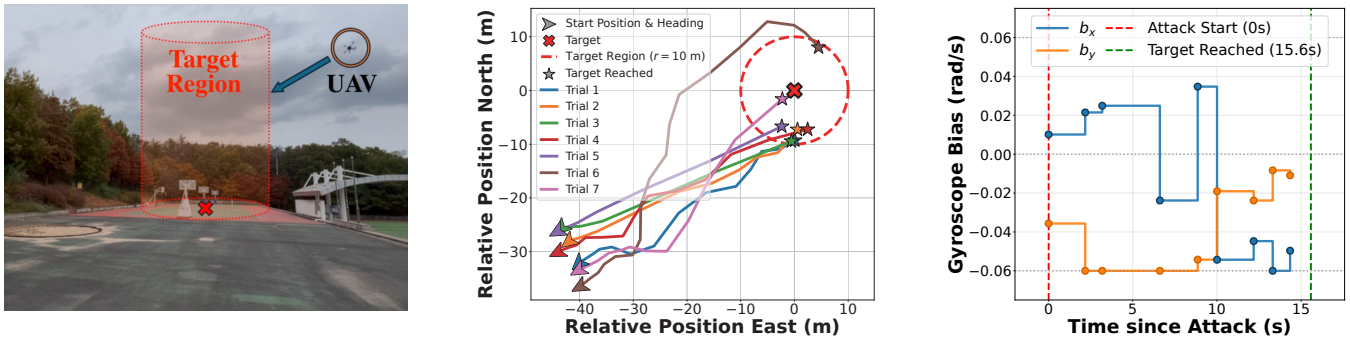
Policy Adaptation. Our threat model (§3.3) assumes that the attacker has access to external tracking equipment that provides ground-truth kinematic states. Although such equipment and supporting technology are available [3, 15, 17], we did not have access to them during hardware testing. Consequently, we trained and adapted a `sim-to-real` policy that relies solely on onboard sensor data.

A key challenge in using onboard sensor data in place of observation is that the UAV's fused state estimates are corrupted by the injected gyroscope bias. To address this, we *reconstruct* observations from raw IMU and GPS readings to approximate the underlying physical states before the fusion algorithm uses them. Specifically:

- **Position and Velocity:** We replace ground-truth values with raw GPS data, deriving horizontal velocity from ground speed and course-over-ground, and vertical velocity from altitude differences across GPS updates.
- **Angular Velocity:** Since the IMU readings include the injected bias, our controller module subtracts the injected bias to reconstruct the true angular rates before feeding them to the policy.
- **Attitude:** Although attitude estimates are affected by the attack-induced estimator drift, no direct raw sensor measurement is available. Thus, we use the fused attitude as an observation despite its gradual corruption. To compensate for the corruption, we augment the actor's partial state vector $\tilde{o}_t$ with linear acceleration and horizontal distance to target. The critic's privileged state vector o_t is similarly extended with 24-dimensional ground truth kinematic states.

The adapted `sim-to-real` policy was trained with over 36K iterations on a 10 m cylindrical objective.

Real-World Experiment Setup. We used Holybro quadcopters with Pixhawk 6C flight controller running PX4: an X500 airframe and an S500 airframe. We conducted a total of seven real-world flight trials across four days to evaluate the attack under varying environmental conditions. In each trial, the UAV hovered at an initial position at 10 m above the



(a) Field experiment photo

(b) Flight trajectories

(c) Injected bias sequence

Figure 8: Real-world flight attack data. A physical UAV running PX4 was attacked in seven independent trials.

ground, and the cylindrical target region was approximately 52 m away, as shown in Figure 8(a) and Figure 8(b). During the attack, the UAV was tethered to a ground anchor with a 50 m safety rope, reaching a maximum displacement of 37 m so the rope remained slack and did not influence the trajectory.

As in simulation, gyroscope bias was injected into the target UAV through a MAVLink message over a telemetry radio link between the UAV and the attacker's ground control station. As detailed in §4.5, in real deployments, this role would be fulfilled by physical injection channels such as acoustic [12, 45] or laser-based [25] methods; MAVLink serves here solely as an interface for controlled experiments.

The field experiment was conducted under dynamic weather conditions (Figure 8(a)), with average wind speeds of 1.7 m/s to 4.4 m/s and gusts up to 7.9 m/s. The onboard GPS provided updates at an average rate of 8 Hz, receiving 26 to 29 satellites during the experiment.

Results. We deployed the `sim-to-real` policy to evaluate our attack on a physical UAV and repeated the experiment seven times, as summarized in Table 6. In *every* trial, the attack successfully redirected the UAV into the 10 m cylindrical objective, following the respective trajectories shown in Figure 8(b). On average, the UAV reached the target in 13.7 s with a trajectory length of 53.6 m.

Figure 8(c) shows bias injection logs from the first trial. The policy initiated with a sustained injection phase lasting 4 s to 10 s, injecting bias of roughly -0.01 rad/s to 0.04 rad/s on the x-axis and -0.06 rad/s on the y-axis. Then, it transitioned to dynamic modulation for trajectory refinement. Although magnitudes vary across trials due to real-world factors, the overall injection patterns were consistent.

Table 6: Summary of seven real-world flight trials.

Trial	Airframe	Init. Dist. (m)	Time (s)	Traj. Len. (m)	Wind (m/s)	Gusts (m/s)
1	X500	51.4	16.0	49.9	1.7 to 2.1	2.8 to 3.1
2	S500	50.5	12.0	48.3	1.9 to 2.8	2.8 to 3.2
3	S500	50.4	8.0	46.0	3.4 to 3.9	4.5 to 6.4
4	S500	53.3	15.0	54.7	3.4 to 4.4	4.7 to 7.9
5	S500	51.3	9.0	46.2	3.4 to 4.4	4.7 to 7.9
6	S500	54.1	22.0	76.5	3.4 to 4.4	4.7 to 7.9
7	S500	52.3	13.9	53.6	3.4 to 4.4	4.7 to 7.9

Robustness Analysis. Our real-world attacks demonstrated robustness against four key real-world factors:

- **Aerodynamic Disturbances:** The experiments were performed over four days with wind speeds ranging 1.7 m/s to 4.4 m/s and gusts up to 7.9 m/s. Our attack policy successfully steered the UAV across all trials, even while the flight controller actively compensated for wind gusts.
- **Airframe Variation:** The Holybro X500 airframe was used in the first trial, whereas the S500, which has a different frame geometry and weight distribution, was used in the other six. The policy succeeded on both airframes without retraining, demonstrating the simulation-based cross-platform results in RQ3 (§7.3).
- **Network Jitter:** Telemetry latency caused irregular injection intervals (Figure 8(c)): across seven flight logs, 87.5 % averaged 1.087 s ($\sigma = 0.076$ s) while 12.5 % were longer due to command losses. The policy nonetheless maintained coherent control. This aligns with the finding in RQ2 (§7.2).
- **Failsafe Resilience:** The GPS enabled the EKF to detect position anomalies, yet no position or velocity failsafe was triggered in any of the flights. The dedicated gyroscope bias failsafe was likewise never activated, again demonstrating resilience to failsafe mechanisms (RQ5, §7.5).

8 Discussion

Our work has a few limitations. First, although our seven real-world flights across four days in §7.6 successfully steered the UAV into the target region, they were performed at distances of approximately 52 m. Validating the policy at longer ranges remains important future work. Second, our attack assumes access to controlled gyroscope-bias injection and external state observation. Although these capabilities lie beyond casual misuse, they are realistic for well-resourced adversaries tar-

geting high-value autonomous UAVs. End-to-end deployment combining injection hardware (*e.g.*, acoustic or laser emitters) with a tracking device would introduce imperfections beyond our simulation, such as residual bias and distance-dependent tracking errors. Examples include residual bias from hardware imprecision and tracking errors that scale with distance and motion. RQ2 (§7.2) provides a first-order assessment by emulating injection-side delays and losses from our real-world flight logs and tracker noise from a published vision tracker [17], under which GAP retained over 87 % success on the 10 m cylinder. A full hardware-in-the-loop characterization remains future work. Third, our policy is optimized for reliably reaching the target region rather than fine-grained terminal behavior, occasionally causing overshoot after arrival. Refining terminal control, such as adjusting bias modulation near the target or incorporating explicit stopping policies, remains future work. Finally, hyperparameter tuning of our RL framework remains future work, as we prioritized demonstrating this previously unexplored attack vector.

Ethics and Disclosure. We disclosed our findings to the PX4 and ArduPilot maintainers and are awaiting their feedback regarding its architectural nature and software-level mitigations.

9 Related Work

Attacks on Reference Sensors. Attacks targeting external reference sensors, such as GPS and magnetometers, have been extensively studied. GPS spoofing [24,31,47] attempts to redirect UAVs by injecting false satellite signals. However, these attacks are difficult to execute; modern sensor fusion algorithms can detect inconsistencies between abrupt GPS shifts and inertial measurements [7,33], and simply injecting spoofing signals triggers receiver-level alarms [16,28,42], which activate failsafes and cause immediate landing or return-to-launch responses, causing hijacking attempts to fail. Similarly, magnetometer false data injection attacks [5] can manipulate heading estimates, but they typically require GPS-denied environments where failsafes are disabled, limiting their operational scope. In contrast, our approach targets the inertial sensors directly, exploiting the fusion architecture to bypass cross-validation mechanisms even when GPS is available.

Attacks on Inertial Sensors. To bypass GPS-layer defenses, recent works have focused on manipulating inertial sensors. Acoustic injection attacks [32,45] exploit MEMS gyroscope resonances to induce drift. Laser-based injection attacks [25] modulate light to induce mechanical vibration that excites MEMS gyroscope resonances, enabling remote signal injection. While stealthy, these physical-layer attacks lack feedback for precise trajectory tracking, resulting in uncontrolled, coarse drifts rather than targeted steering. ConfuSense [10] pioneers RL-driven UAV attacks by using IEMI-based sensor reconfiguration to suspend or slow IMU sampling. Its policy learns when to activate interference (binary timing), leading to coarse directional deviation (*e.g.*, left/right drift). In contrast, GAP preserves the gyroscope stream and injects continuous two-axis bias, learning a closed-loop control policy over bias values. This shifts the attack from discrete disruption to continuous control of the estimation dynamics. As a result, GAP enables precise target-region steering at non-trivial distances by exploiting estimator–controller lag, going beyond coarse directional deviation (§7.1).

Defenses and Evasion. Various physics-based detectors [6, 13, 29] identify sensor attacks by monitoring violations in control-theoretic relationships between control commands and sensor responses. However, recent studies have shown that these defenses rely on static thresholds or fixed invariant models [9]. Our evaluation shows the attack achieves a 76 % evasion rate against threshold-based defenses (*e.g.*, CI [6]), not by learning evasion strategies, but because the bias magnitude, constrained to bypass the EKF's internal failsafe, inherently falls within the detection blind spots of such detectors. Recovery-oriented defenses such as SpecGuard [8] achieve strong recovery success across a broad range of physical attacks by using RL to keep UAVs compliant with mission specifications, running Proactive Control (PC) throughout the flight and engaging Reactive Control (RC) after attack detection. However, two factors suggest GAP would likely remain effective against both. First, GAP keeps bias below PX4's gyroscope failsafe and triggers other failsafes or the CI detector in only about 20 % of runs (§7.4, §7.5), so RC is unlikely to engage in most executions. Second, the publicly available SpecGuard implementation models gyroscope attacks by modifying EKF attitude estimates instead of sensor-driver-level readings. As a result, it evaluates only the Slow Path effect (§3.1) of gyroscope spoofing and not the Fast Path that GAP exploits. We leave empirical validation of GAP against SpecGuard-PC/RC in this setting as future work.

10 Conclusion

We identify an architectural vulnerability in the UAV control pipeline: high-rate control loops operating atop low-rate, indirectly fused state estimates. This temporal mismatch opens an exploitable window in which carefully crafted gyroscope bias injections can steer a UAV to an arbitrary target region, while remaining undetected by a state-of-the-art detector and being robust to built-in failsafes. Leveraging this insight, we developed a reinforcement learning-based black-box attack policy that exploits this window under assumed bias-injection and external observation capabilities. Our trained policy, GAP, achieved 85 % success in target-region steering and bypassed a detector in 76 % of successful attack cases. We validated its real-world feasibility, successfully steering a physical UAV into the target region under varying winds and airframes.

11 Acknowledgment

We would like to thank our shepherd and the anonymous reviewers for their insightful comments and constructive feedback that helped improve this paper. We are also grateful to Jangseop Choi, Junwoong Doh, Chiheon Kim, and Minki Lee from the POSTECH CompSec Lab for their field setup assistance with the physical drone experiments. This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-23523174), and the Institute of Information & Communications Technology Planning & Evaluation (IITP) - ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT) (IITP-2026-RS-2024-00437866 and IITP-2026-RS-2026-25529760).

References

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] Nourdine Aliane. A survey of open-source uav autopilots. *Electronics*, 13(23):4785, 2024.

[3] Nancy Alshaer, Reham Abdelfatah, Tawfik Ismail, and Haitham Mahmoud. Vision-Based UAV Detection and Tracking Using Deep Learning and Kalman Filter. *Computational Intelligence*, 41(1):e70026, 2025.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[5] Wenxin Chen, Yingfei Dong, and Zhenhai Duan. Manipulating drone dynamic state estimation to compromise navigation. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2018.

[6] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 801–816, 2018.

[7] Sagar Dasgupta, Mizanur Rahman, Mhafuzul Islam, and Mashrur Chowdhury. A sensor fusion-based GNSS spoofing attack detection framework for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):23559–23572, 2022.

[8] Pritam Dash, Ethan Chan, and Karthik Pattabiraman. Specguard: Specification aware recovery for robotic autonomous vehicles from physical attacks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1849–1863, 2024.

[9] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. Stealthy attacks against robotic vehicles protected by control-based intrusion detection techniques. *Digital Threats: Research and Practice*, 2(1):1–25, 2021.

[10] Alessandro Erba, John H Castellanos, Sahil Sihag, Saman Zonouz, and Nils Ole Tippenhauer. ConfuSense: Sensor Reconfiguration Attacks for Stealthy UAV Manipulation. In *3rd USENIX Symposium on Vehicle Security and Privacy (VehicleSec 25)*, pages 45–62, 2025.

[11] Center for Strategic and International Studies. How Ukraine's Operation Spider's Web Redefines Asymmetric Warfare, June 2025. https://www.csis.org/analysis/how-ukraines-spider-web-operation-redefines-asymmetric-warfare. Accessed: 2025-11-11.

[12] Ming Gao, Lingfeng Zhang, Leming Shen, Xiang Zou, Jinsong Han, Feng Lin, and Kui Ren. Exploring practical acoustic transduction attacks on inertial sensors in MDOF systems. *IEEE Transactions on Mobile Computing*, 23(5):3539–3557, 2023.

[13] Jairo Giraldo, David Urbina, Alvaro Cardenas, Junia Valente, Mustafa Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. A survey of physics-based attack detection in cyber-physical systems. *ACM Computing Surveys (CSUR)*, 51(4):1–36, 2018.

[14] Matthew J Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI fall symposia*, volume 45, page 141, 2015.

[15] Brian KS Isaac-Medina, Matt Poyser, Daniel Organisciak, Chris G Willcocks, Toby P Breckon, and Hubert PH Shum. Unmanned aerial vehicle visual detection and tracking using deep neural networks: A performance benchmark. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1223–1232, 2021.

[16] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. Unmanned aircraft capture and control via GPS spoofing. *Journal of field robotics*, 31(4):617–636, 2014.

[17] Chujun Li, Yiyang Wu, Sheng Zhuge, Xia Yang, Bin Lin, Xiangpeng Xu, and Xiaohu Zhang. Robust 6D

Pose Estimation of the UAV Based on Hybrid Features. *IEEE Transactions on Instrumentation and Measurement*, 2024.

[18] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

[19] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[20] Michael Luo, Jiahao Yao, Richard Liaw, Eric Liang, and Ion Stoica. Impact: Importance weighted asynchronous architectures with clipped target networks. *arXiv preprint arXiv:1912.00167*, 2019.

[21] ModalAI, Inc. ModalAI Launches VOXL 2, the Smallest, Smartest Blue UAS Framework Autopilot, 2022. https://www.businesswire.com/news/home/20220425005133/en/. Accessed: 2025-12-05.

[22] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[23] Shoei Nashimoto, Daisuke Suzuki, Takeshi Sugawara, and Kazuo Sakiyama. Sensor con-fusion: Defeating kalman filter in signal injection attack. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 511–524, 2018.

[24] Juhwan Noh, Yujin Kwon, Yunmok Son, Hocheol Shin, Dohyun Kim, Jaeyeong Choi, and Yongdae Kim. Tractor beam: Safe-hijacking of consumer drones with adaptive GPS spoofing. *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–26, 2019.

[25] Kunihiko OI and Takeshi SUGAWARA. Signal Injection Attack on Inertial Measurement Unit using Continuous-Wave Laser. *IEICE Transactions on Information and Systems*, 2025.

[26] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[27] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[28] Mark L Psiaki and Todd E Humphreys. GNSS spoofing and detection. *Proceedings of the IEEE*, 104(6):1258–1270, 2016.

[29] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. SAVIOR: Securing autonomous vehicles with robust physical invariants. In *29th USENIX security symposium (USENIX Security 20)*, pages 895–912, 2020.

[30] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.

[31] Harshad Sathaye, Martin Strohmeier, Vincent Lenders, and Aanjhan Ranganathan. An experimental study of GPS spoofing and takeover attacks on UAVs. In *31st USENIX security symposium (USENIX security 22)*, pages 3503–3520, 2022.

[32] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. Rocking drones with intentional sound noise on gyroscopic sensors. In *24th USENIX security symposium (USENIX Security 15)*, pages 881–896, 2015.

[33] Çağatay Tanıl, Samer Khanafseh, Mathieu Joerger, and Boris Pervan. Kalman filter-based INS monitor to detect GNSS spoofers capable of tracking aircraft position. In *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 1027–1034. IEEE, 2016.

[34] ArduPilot Development Team. Code Overview (Copter), 2025. https://ardupilot.org/dev/docs/apmcopter-code-overview.html. Accessed: 2025-12-05.

[35] ArduPilot Development Team. Crop Sprayer, 2025. https://ardupilot.org/copter/docs/common-sprayer.html. Accessed: 2026-05-12.

[36] ArduPilot Development Team. ArduPilot Documentation, 2026. https://ardupilot.org. Accessed: 2026-04-27.

[37] ArduPilot Development Team. Identification of a Multicopter, 2026. https://ardupilot.org/copter/docs/systemid-model-development.html. Accessed: 2026-04-27.

[38] PX4 Development Team. PX4 Architectural Overview, 2025. https://docs.px4.io/main/en/concept/architecture. Accessed: 2025-12-05.

[39] PX4 Development Team. PX4 Autopilot Documentation, 2025. https://docs.px4.io/main/en/. Accessed: 2025-11-29.

[40] PX4 Development Team. PX4 Commercial Systems, 2025. https://px4.io/ecosystem/commercial-systems/. Accessed: 2025-12-05.

[41] PX4 Development Team. PX4 Controller Diagrams, 2026. https://docs.px4.io/main/en/flight_stack/controller_diagrams. Accessed: 2026-04-27.

[42] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful GPS spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86, 2011.

[43] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[44] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. WALNUT: Waging doubt on the integrity of MEMS accelerometers with acoustic injection attacks. In *2017 IEEE European symposium on security and privacy (EuroS&P)*, pages 3–18. IEEE, 2017.

[45] Yazhou Tu, Zhiqiang Lin, Insup Lee, and Xiali Hei. Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors. In *27th USENIX security symposium (USENIX Security 18)*, pages 1545–1562, 2018.

[46] Zhanghao Wu, Eric Liang, Michael Luo, Sven Mika, Joseph E Gonzalez, and Ion Stoica. RLlib flow: Distributed reinforcement learning is a dataflow problem. In *Conference on Neural Information Processing Systems (NeurIPS)*, page 54, 2021.

[47] Kexiong Curtis Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. All your GPS are belong to us: Towards stealthy manipulation of road navigation systems. In *27th USENIX security symposium (USENIX security 18)*, pages 1527–1544, 2018.

USENIX VehicleSec '26 Artifact Appendix: Blind Spot in Sensor Fusion: Exploiting an Architectural Vulnerability to Hijack and Precisely Control UAVs

Jongsoo Han
Pohang University of Science and Technology
hanjs@postech.ac.kr

Seulbae Kim
Pohang University of Science and Technology
seulbae@postech.ac.kr

A Artifact Appendix

A.1 Abstract

This artifact accompanies the paper *Blind Spot in Sensor Fusion: Exploiting an Architectural Vulnerability to Hijack and Precisely Control UAVs*. It contains patched PX4/ArduPilot firmware, trained GAP policies, experiment wrappers, analysis scripts, and pre-baked outputs. It supports six claims on attack effectiveness, robustness to tracking noise and delay/loss, cross-platform transfer, CI-detector evasion, failsafe analysis, and sim-to-real feasibility. The recommended evaluator path uses the prepared VM or archived source snapshot to run the smoke test and verify the shipped pre-baked claims. Optional fresh reruns are provided as additional experiments.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The normal evaluation path is software-only. It uses either shipped pre-baked logs or software-in-the-loop evaluation, and does not require disabling host security features. The optional CI-detector rerun uses a legacy VMware image that should be treated as an isolated guest environment. The optional real-flight rerun uses a physical UAV and should only be attempted with appropriate permissions, a safe outdoor flight site, and independent flight-safety procedures.

A.2.2 How to access

The stable archival copy of the artifact is available through Zenodo: https://doi.org/10.5281/zenodo.19652756. The archive provides two entry points: (i) a prepared Ubuntu 22.04 VirtualBox image with the repository, Python environment, built firmware, QGroundControl, policies, and pre-baked outputs already installed, and (ii) a versioned source snapshot that can be built from scratch with `./setup.sh`. The Zenodo bundle also includes the trained model archive, the optional legacy VMware image for CI-detector reruns, and the QGroundControl AppImage for source-based users.

A.2.3 Badges

This artifact was awarded the following badges:

- **Artifacts Available**
- **Artifacts Functional**
- **Results Reproduced**

A.2.4 Hardware dependencies

The main path requires a recent `x86_64` host with 12 CPU cores, 16GB RAM, and no GPU. A 100GB disk is recommended, as `./setup.sh` increased disk usage by about 30GB on our Ubuntu 22.04 guest. Optional physical reruns require a multicopter, Pixhawk, telemetry radios, and a safe flight site.

A.2.5 Software dependencies

The prepared VM already includes all dependencies. Source-based users need Ubuntu 22.04, Python 3.10, Java for jMAVSim, Ray/RLlib, NumPy, pandas, matplotlib, scipy, pyulog, geographiclib, the patched PX4 and ArduPilot source code, and the custom MAVLink support installed by `./setup.sh`. VirtualBox is needed for the provided VM, while VMware and QGroundControl are needed only for the optional CI-detector and real-flight workflows, respectively.

A.2.6 Benchmarks

The artifact evaluates the shipped trained GAP and sim-to-real policies and the following result sets under `results/pre-baked/`:

- shipped pre-baked results
- PX4 attack-flight CSVs under `flight-logs/`
- real-flight ULog files under `rq6/`

A.3 Set-up

A.3.1 Installation

Path A is recommended: download the prepared GAP VM from Zenodo, import it into VirtualBox, log in with user/password `gap`/`gap`, and open a terminal at `$HOME/GAP`. All

dependencies, built firmware, trained policies, and QGroundControl are already installed.

Path B uses the archived source snapshot or GitHub repository. From `$HOME/GAP`, run:

```
./setup.sh
source .venv/bin/activate
```

The setup script installs dependencies, compiles patched PX4 and ArduPilot, creates a local Python environment, and took about 30 minutes on our 12-core Ubuntu 22.04 host. If starting from GitHub rather than the Zenodo snapshot, download `gap-models.zip` from Zenodo and unpack it into `src/gap/models/` as documented in the repository README.

A.3.2 Basic Test

Run:

```
./experiments/run_smoke_test.sh
```

Expected outcome: `[PASS] All 11 checks passed`. This test checks the environment and verifies the presence and consistency of the shipped data without running fresh experiments.

A.4 Evaluation Workflow

A.4.1 Major claims

(C1) GAP outperforms the three baselines in PX4-jMAVSim and reaches the 10 m cylindrical target region with an 85.1 % success rate.

(C2) GAP remains effective under tracking noise, injection delay/loss, and their combination.

(C3) The trained policy transfers without retraining to PX4 Gazebo and nine ArduPilot airframes.

(C4) GAP bypasses the CI detector in most successful attacks. In the shipped RQ4 set, 21 of 24 attacks succeed and only 5 of the successful attacks trigger the detector.

(C5) Built-in failsafes do not reliably stop the attack. In the shipped PX4 attack-flight corpus, the gyroscope-bias failsafe never triggers, while generic failsafes appear in 213 of 1,048 successful attacks.

(C6) The sim-to-real policy succeeds in all seven shipped real-world flight trials, reaching the 10 m cylindrical target in 13.794 s on average with a mean trajectory length of 53.576 m.

A.4.2 Experiments

(E0) Shipped reference path [5 human-minutes + 10 compute-minutes]. Run:

```
bash analysis/verify_claims.sh pre-baked
bash analysis/generate_all.sh pre-baked
```

This verifies the claims from the pre-baked data, regenerates all shipped tables and figures, and supports C1-C6.

(E1) RQ4 CI-detector analysis [5 human-minutes + 5 compute-minutes]. Run:

```
python3 -m analysis.generate_rq4_analysis \
--source pre-baked
```

This regenerates the RQ4 CI-detector evasion CSV and figure from shipped JSON logs and supports C4.

(E2) RQ5 failsafe analysis [5 human-minutes + 5 compute-minutes]. Run:

```
python3 -m analysis.generate_rq5_analysis \
--source pre-baked
```

This post-processes the shipped PX4 attack-flight corpus to compute failsafe activation statistics and supports C5.

(E3) RQ6 real-flight analysis [5 human-minutes + 5 compute-minutes]. Run:

```
python3 -m analysis.generate_rq6_table6 \
--source pre-baked
python3 -m analysis.generate_rq6_figure8 \
--source pre-baked
```

This regenerates Table 6 and Figure 8 from the shipped real-flight ULog files and supports C6.

For all experiments, expected outputs are CSV files under `analysis/csv/` and figures under `analysis/figures/`.

A.5 Additional Experiments - Fresh Runs

Fresh reruns of RQ1-RQ3 are provided as additional experiments for exercising the simulator execution path. They are stochastic and are intended as directional validation, while the shipped pre-baked data remains the exact paper result set. The one-command fresh path is:

```
./experiments/run_all.sh --mode approx
```

E4 reruns RQ1 and regenerates Table 3/Figure 7. E5 reruns RQ2 and regenerates Table 4. E6 reruns RQ3 and regenerates Table 5. Detailed commands and optional CI-detector and real-flight reruns are documented in the artifact repository.

A.6 Notes on Reusability

The repository separates reusable evaluation code under `src/evaluation/` from thin experiment wrappers under `experiments/`. The same analysis scripts can be run on shipped pre-baked data or user-generated fresh data, and outputs are source-tagged to avoid overwriting one another. The artifact covers evaluation of the shipped trained policies. The training pipeline is out of scope.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20260101. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/vehiclesec2026/.